NANO SYSTEMS CORP.®

REFERENCE CARD FOR THE

ATARI® 400/800™ MICROCOMPUTERS

WITH ATARI BASIC AND MICROSOFT® BASIC

ATARI is a registered Trademark of Atari, Inc.
400/800 is a Trademark of ATARI, INC.
MICROSOFT is a Registered Trademark of MICROSOFT Corp.
© 1983, PHILIP R. FASSNACHT, and, SHIRLEY A. & PAUL P. NANOS

SPECIAL CHARACTERS AND OPERATORS

☆ = ATARI BASIC ONLY ★ = MICROSOFT BASIC ONLY

blank	End of BASIC stmt. Tab to next line after PRINT.
?	Use this instead of keying in PRINT.
,	Tab to next tab after PRINT.
;	Suppress tab after PRINT.
:	End of BASIC stmt. Tab to next line after PRINT.
· space	☆Use this instead of REM.
!	★Use this instead of REM.
'	★Use this instead of REM.
" "	String delimiter (quotes).
""	★Double quote pair inside of quotes gives string of quote.
$	Identifies string variable. (A$ = "data")
null	☆Identifier not used for ATARI numeric variable. (A=23.5)
null	★Identifier not used for MSOFT single precision variable.
#	★Identifier of double-precision variable (A#=23)
%	★Identifier of integer variable. (A%=23)
&	★Hexadecimal notation. (&7EF)
( )	Top priority in order of operations. ((3*4)/ 2)
^	★Exponentiation. Power of. (A ^ B)
−	Minus. (−23)
+	Plus. (+23)
+	★Concatenation. (A$ + B$)
*	Multiplication. (A * B)
/	Division. (A / B)
+	Addition. (A + B)
−	Subtraction. (A − B)
=	Equals (IF A = B) −or− assignment (A = 23)
<>	Not equal. (IF A <> B)
<	Less than. (IF A < B)
>	Greater than. (IF A > B)
<=	Less than or equal to. Not greater than. (IF A <= B)
>=	Greater than or equal to. Not less than. (IF A >= B)
NOT	Reverses true/ false result. (IF A NOT >= B)
AND	If both expr are true, result is true. (IF exp AND exp)
OR	If either or both expr are true, result is true. (IF exp OR exp)
XOR	★If one expr is true and one expr is false, result is true. (IF expr XOR exp)
1	True.
0	False.
Non-zero	★True.

ATARI BASIC ABBREVIATIONS — STMT (ABBREV.)

BYE (B.)	ENTER (E.)	LPRINT (LP.)	REM (R.) (.△)
CLOAD (CLOA.)	FOR (F.)	NEXT (N.)	RESTORE (RES.)
CLOSE (CL.)	GET (GE.)	NOTE (NO.)	RETURN (RET.)
COLOR (C.)	GOSUB (GOS.)	OPEN (O.)	RUN (RU.)
CONT (CON.)	GOTO (G.)	PLOT (PL.)	SAVE (S.)
CSAVE (CS.)	GRAPHICS (GR.)	POINT (P.)	SETCOLOR (SE.)
DATA (D.)	INPUT (I.)	POKE (POK.)	SOUND (SO.)
DEG (DE.)	LET (LE.)	POSITION (POS.)	STATUS (ST.)
DIM (DI.)	LIST (L.)	PRINT (PR.) (?)	STOP (STO.)
DOS (DO.)	LOAD (LO.)	PUT (PU.)	TRAP (T.)
DRAWTO (DR.)	LOCATE (LOC.)	READ (REA.)	XIO (X.)

SYSTEM COMMANDS

☆ = ATARI BASIC ONLY ★ = MICROSOFT BASIC ONLY

FORMAT • *DESCRIPTION*

★**AUTO** [num, incr] • *Begin auto-num at num using incr increments.*
☆**BYE** • *Exit BASIC, enter MEMOPAD mode.*
CLOAD • *Load CSAVEd BASIC pgm into memory from cassette.*
CONT • *CONTinue execution after STOP, BREAK, or END.*
CSAVE • *Save BASIC pgm onto cassette from memory.*
★**DEL** [n1 [– [n2] • *Delete BASIC lines n1 thru n2.*
DOS • *Exit BASIC, enter Disk Operating System.*
☆**ENTER** "**filespec**" • *Merge LISTed pgm into memory from tape/ disk.*
★**KILL** "**filespec**" • *Delete program file from disk.*
☆**LIST** [n1 [,n2] • *Display BASIC lines n1 thru n2.*
☆**LIST** "**P:**" [,n1 [,n2] • *Print BASIC lines n1 thru n2 on printer.*
☆**LIST** "**filespec**" [,n1 [,n2] • *Save BASIC lines n1 thru n2 on device.*
★**LIST** [n1 [–n2] • *Display BASIC lines n1 thru n2.*
★**LIST** "**P:**" [n1 [–n2] • *Print BASIC lines n1 thru n2 on printer.*
★**LIST** "**filespec**" [n1 [–n2] • *Save BASIC lines n1 thru n2 on device.*
LOAD "**filespec**" • *Load pgm into memory from device.*
★**LOCK** "**filespec**" • *Install write-protect for file. (See XIO)*
★**MERGE** "**filespec**" • *Merge LISTed pgm into memory from tape/ disk.*
★**NAME** "**filespec**" **TO** "**fname.ext**" • *Rename a disk file.*
NEW • *Clear program from memory.*
★**RENUM** [n1 [,n2 [,n3] • *Renumber BASIC lines n2 thru end starting with number n1, using increment n3.*
RUN • *Execute BASIC program.*
RUN "**filespec**" • *Load and execute BASIC program.*
★**RUN** [lineno] • *Execute BASIC program beginning with lineno.*
☆**SAVE** "**filespec**" • *Save BASIC pgm from memory to tape/ disk.*
★**SAVE** "**filespec**" [**LOCK**] • *Save BASIC pgm with write-protect.*
★**TROFF** • *Exit trace mode.*
★**TRON** • *Enter trace mode. (Displays each line num executed).*
★**UNLOCK** "**filespec**" • *Remove write-protect from file. (See XIO)*
★**VERIFY** "**filespec**" • *Compare memory pgm against tape/ disk pgm.*
☆**XIO 35,#devno,0,0**"**filespec**" • *Install file write-protect. (See LOCK)*
☆**XIO 36,#devno,0,0,**"**filespec**" • *Remove file write-protect. (See UNLOCK)*
☆**XIO 254,#devno,0,0,**"**dev**" • *Clear and initialize disk.*

filespec = **devicecode:filename.ext**

BASIC STATEMENTS

☆ = ATARI BASIC ONLY ★ = MICROSOFT BASIC ONLY

STATEMENT • *DESCRIPTION*

PROGRAM CONTROL

★**AFTER** (num) [**GOTO**] lineno • **GOTO** lineno **AFTER** *num time. num=1/60th secs.*
END • *End program execution.*
★**ERROR** code • *Force an error to occur.*
FOR var=n1 **TO** n2 [**STEP** incr] • *Loop thru* **NEXT** *until var greater n2.*
GOSUB lineno • *Perform routine at lineno until* **RETURN**.
GOTO lineno • *Go execute lineno next. (Branch or jump to lineno).*
IF cond **THEN** lineno • *Goto lineno if condition met.*
★**IF** cond **THEN** lineno **ELSE** lineno • *Goto lineno based upon condition.*
IF cond **THEN** action • *Execute action if condition met.*
★**IF** cond **THEN** action1 **ELSE** action2 • *Execute action based on condition.*
☆**NEXT** var • *Indicate end of* **FOR** *loop.*
★**NEXT** [var] • *Indicate end of* **FOR** *loop.*
ON num **GOSUB** lineno [,lineno . . .] • *Perform routine num in lineno list.*
ON num **GOTO** lineno [,lineno . . .] • *Goto line num in lineno list.*
★**ON ERROR** lineno • *Goto lineno if error occurs.*
★**ON ERROR GOTO 0** • *Deactivate the* **ON ERROR** *lineno statement.*
★**RESUME** [lineno [**NEXT**] • *Return from* **ON ERROR** *routine.*
RETURN • **RETURN** from a **GOSUB**.
STOP • *Halt program execution until* **CONT**.
☆**TRAP** lineno • *Goto lineno if error occurs.*
★**WAIT** addr,byte1,byte2 • *Pause until* **PEEK**(*addr*)=*byte1* **AND**ed *to byte2.*

MEMORY/ DATA

☆ = ATARI BASIC ONLY ★ = MICROSOFT BASIC ONLY

★CLEAR • *Reset all vars,* DIMs, *and* DATA *pointer.*
★CLEAR STACK • *Cancel time counter. Abort the* AFTER *statement.*
☆CLR • *Reset all vars,* DIMs, *and* DATA *pointer.*
★COMMON var [,var . . .] • *Keep var value in pgm2 same as in pgm1.*
★COMMON ALL • *Keep all var values in pgm2 same as in pgm1.*
☆COM svar(num) [,svar(. . .] • *Define string variable length array.*
☆COM numvar(n1 [,n2]) [,numvar(. . .] • *Define numeric array/ matrix table.*
DATA data [,data . . .] • *Store data into memory table for later* READ.
★DEF userfunct(arg [,arg])=formula • *Define user function.*
★DEFDBL char [,char,char-char. . .] • *Define double precision var prefixes.*
★DEFINT char [,char,char-char. . .] • *Define integer variable prefixes.*
★DEFSNG char [,char,char-char . . .] • *Define single precision var prefixes.*
★DEFSTR char [,char,char-char . . .] • *Define string variable prefixes.*
DIM svar(num [,svar(. . .] • *Define string variable length array.*
DIM numvar(n1 [,n2]) [,numvar(. . .] • *Define numeric array/ matrix table.*
★DIM var(n1 [,n2,n3 . . .]) [,*var*(. . .] • *Define array/ matrix table.*
[LET] var=value • *Assign a value to a variable.*
★MOVE fromaddr,toaddr,length • MOVE *memory data from one loc to another.*
★OPTION BASE num • *Set* DIM *starting subscript to* 0 *or* 1. *Default = 0.*
★OPTION CHRnum • *Reserve memory for character data. (0-2).*
0 = Release All, 1 = 1024 bytes, 2 = 512 bytes.
★OPTION RESERVE num • *Reserve num bytes of memory for desired purpose.*
POKE addr,num • *Store num at addr.*
☆POP • *Clear top of stack. (Use if no* RETURN *or* NEXT *after* GOSUB *or* FOR*).*
★RANDOMIZE [num] • *Re-seed random number generator.*
READ var [,var . . .] • READ *data from* DATA *table.*
☆:REM -or- period space(.) • *Remarks follow this. (Ignored by computer).*
★:REM -or- ! -or- ' • *Remarks follow this. (Ignored by computer).*
RESTORE [lineno] • RESTORE DATA *pointer (default=first* DATA *line).*
★TIME$ = "hh:mm:ss" • *Set clock to hours, minutes, seconds.*

GRAPHICS

☆ = ATARI BASIC ONLY ★ = MICROSOFT BASIC ONLY

★CLS [color+lum] • *Clear graphics screen to* color *and* lum.
COLOR reg • *Specify color register to be used for* PLOT/ DRAWTO.
(If GRAPHICS *modes 0-2, specifies a character)*
☆DRAWTO col,row • *Draw line from last* PLOT *to* col,row *using char or color specified by* COLOR *reg.*
★FILL col,row TO col,row • *Fill w/ char or color specified by* COLOR *reg. (Bound. = last* PLOT *line(s).*
GRAPHICS mode • *Select screen mode. (0-11,16-27,32-43,48-59).*
☆LOCATE col,row,numvar • *Load screen data at* col,row *into* numvar.
PLOT col,row • PLOT *point at* col,row *using char or color specified by* COLOR *reg.*
★PLOT col,row [TO col,row [TO . . .] • PLOT point or line(s) using color specified by COLOR *reg.*
☆POSITION col,row • *Move invisible cursor to* col,row *on screen.*
☆SETCOLOR reg,color,lum • *Load color* reg *(0-4) with* color *(0-15) and* lum/ *brightness (0-15).*
★SETCOLOR reg,color,lum • *Load color* reg *(0-8) with* color *(0-15) and* lum/*brightness (0-15).*
(Regs 0-3=Player/ missile color regs)
(Regs 4-8=Normal. Same as ATARI *regs 0-4.)*
☆XIO 18,#iocb,0,0,"S:" • *Fill w/ color from last* PLOT/ POSITION *to* DRAWTO *lines.* COLOR = POKE 765,regno (1-3).

PLAYER-MISSILE GRAPHICS

☆ = ATARI BASIC ONLY ★ = MICROSOFT BASIC ONLY

★numvar = VARPTR (PLM1) • *Get addr of single-line res* P/ M *area.*
★numvar = VARPTR (PLM2) • *Get addr of double-line* P/ M *area.*
★MOVE top-addr,bot-addr,bytecount • *Move player up or down.*
★OPTION PLMnum • *Zero and reserve memory for* P/ M *graphics.*
PLM0 = *Release all* PLM *space.*
PLM1 = *1280 bytes for single-line resolution.*
PLM2 = *640 bytes for double-line resolution.*
★SETCOLOR reg,color,lum • *Load* color P/ M reg *(0-3) with* color *0-15 and* lum *(0-15, brightness).*
(Regs 0-3 = color for P/ M *0-3)*

INPUT/OUTPUT

☆ = ATARI BASIC ONLY ★ = MICROSOFT BASIC ONLY

★AT(x,y) • *Used with* PRINT/INPUT/GET/PUT. *Points to screen loc.*
★AT(sector,num) • *Used with* PRINT/INPUT/GET/PUT. *Points to disk loc.*
(See NOTE *statement —* sector,num)

CLOSE #iocb • *Close a file. (*iocb = *0-7)*

GET #iocb,numvar • *Read byte from device. (*#6 = screen)
★GET #iocb,numvar [,numvar . . .] • *Read byte(s) from device.*
★GET #iocb,AT(sector,num);numvar [,numvar] • *Read byte(s) from disk.*
★GET #6,AT(x,y);numvar [,numvar . . .] • *Read screen byte(s) at* x,y.

INPUT var [,var . . .] • *Read keyed input into var(s).*
INPUT #iocb [,[;] var [,var . . .] • *Read record(s) from device.*
★INPUT "msg";var [,var . . .] • *Prompt & read keyed response.*
★INPUT #iocb "msg"; var [,var . . .] • *Prompt. Read device record(s).*
★INPUT #iocb ["msg"],AT(sector,num);var [,var . . .] • *Prompt, read disk.*
★INPUT #6, ["msg"],AT(x,y);var [,var . . .] • *Prompt, read screen.*

★LINE INPUT [#iocb] ["msg";] $var • *Prompt, read keyboard (raw data).*
☆LOCATE x,y,numvar • *Read data at screen position* x,y.
☆LPRINT [item [; [, [item . . .] • *Print* items *on printer.*
NOTE #iocb,var1,var2 • *Save disk position.* var1=*sector.* var2=*byteno.*

☆OPEN #iocb,code,auxcode,"filespec" • *Open a file for access.*
code = 4 — *Input only*
8 — *Output only*
12 — *Input and output*
6 — *Disk directory input*
9 — *Output write to the back of the file*
auxcode — *Used if needed by device. See its manual.*

★OPEN #iocb,"filespec",opt • *Open a file for access.*
opt = INPUT — *Input only*
OUTPUT — *Output only*
UPDATE — *Input or output*
APPEND — *Output write to back of file*

☆POINT #iocb,var1,var2 • *Position disk to* NOTEd *position.*
☆POSITION x,y • *Move invisible cursor to point* x,y *on screen.*

PRINT [item {;|,} [item . . .] • *Display item(s) on screen.*
PRINT #iocb [item {;|,} [item . . .] • *Write* item(s) *to device.*
★PRINT #6,AT(x,y); [item {;|,} [item . . .] • *Display* item(s) *at* x,y.
SEE SPC, TAB, & USING

PUT #iocb,numvar • *Write a byte to device. (#6 is screen)*
★PUT #iocb,AT(sector,num);numvar • *Write byte to disk.*
★PUT #6,AT(x,y);numvar • *Write byte to screen at position* x,y.

★SPC(num) • *Used with print.* num = *number of spaces desired.*
☆STATUS #iocb, numvar • *Puts device status code into* numvar.
★numvar = STATUS ([iocb [filespec]) • *Puts dev status into* numvar.
★TAB(num) • *Used with print.* num = *column to tab-to.*
★USING "pattern" • *Used with print. Sets data for edit pattern.*
SEE PRINT USING EXAMPLES.
☆XIO cmd,#iocb,ctl1,ctl2,"filespec" • *Execute I/O command specified by* cmd *for device* iocb.
SEE XIO STATEMENT COMMAND CODES PANEL

XIO STATEMENT COMMAND (cmd) CODES

cmd	FUNCTION	DESCRIPTION	BASIC
3	OPEN A FILE	*Prepare a file for access.*	**OPEN**
5	GET A RECORD	*Read a record.*	**INPUT**
7	GET A CHARACTER	*Read a byte*	**GET**
9	PUT A RECORD	*Write a record.*	**PRINT**
11	PUT A CHARACTER	*Write a byte.*	**PUT**
12	CLOSE A FILE	*Close (deactivate) a file.*	**CLOSE**
13	GET I/O STATUS	*Get status code of last I/O operation.*	**STATUS**
17	DRAW A LINE	*Draw a line on the screen.*	**DRAWTO**
18	FILL WITH COLOR	*Fill screen area with color specified.*	**XIO**
32	RENAME A FILE	*Change the name of a file.*	
33	DELETE A FILE	*Erase a file from its device.*	
35	LOCK A FILE	*Prevent access to a file.*	
36	UNLOCK A FILE	*Allow access to a file.*	
37	POINT TO DATA	*Save address of current disk position.*	**POINT**
38	NOTE DATA	*Access POINT position on disk.*	**NOTE**
254	FORMAT A DISK	*Clear and initialize a disk.*	

PRINT USING EXAMPLES

The **USING** function of the **PRINT** statement allows you to specify a pattern which is then used to edit your data field before it is printed.

100 PRINT USING "pattern";item . . . .

Special characters are used in the pattern. They tell the computer how to edit your field. The special characters are noted below.

CHAR	FUNCTION
#	One # required for each digit of a number to be printed. When number smaller than pattern, left positions are blanked. When number larger than pattern, % (pct) precedes printed field.
.	Decimal point. Aligns where you specify in ##.## pattern. If number is whole number, zeros fill to right of decimal. Automatic rounding occurs when num greater than decimal positions.
,	Comma. Causes comma to be automatically inserted every 3 digits.
**	Two asterisks placed at front of pattern count as digit symbols. They cause left blanked positions to be asterisk-filled.
$	Dollar-sign placed at front of pattern causes computer to insert a dollar-sign to the left of the field.
$$	Two dollar-signs placed at front of pattern causes computer to insert a dollar-sign to the left of the first digit.
**$	Two asterisks and a dollar-sign placed at front of pattern causes computer to insert dollar-sign to left of first digit and fill field with asterisks to left of dollar-sign.
^^^^	Four carats behind # symbols cause exponential notation (E or D).
+	Plus sign in front or back prints + for positive, – for negative.
–	Minus sign at back prints blank for positive, "–" for negative.
!	Exclamation point extracts and prints only first char of string.
% %	Percent, blanks, percent – extracts part of a string equal to the size of the two percents plus the blanks in between.

EXAMPLE	RESULTS
PRINT USING "####";2345	2345
PRINT USING "#####"2345	2345
PRINT USING "###";2345	%2345
PRINT USING "#######";2345	2345
PRINT USING "##.##";2345	23.45
PRINT USING "##.#";2345	23.5
PRINT USING "####.##";2345	2,345.00
PRINT USING "**#####.##";2345	****23.45
PRINT USING "$#####.##";2345	$ 23.45
PRINT USING "$$#####.##";2345	$23.45
PRINT USING "** $#######.##";2345	*** $23.45
PRINT USING "##^^^^ ";500	5E+02
PRINT USING "###";500	+500
PRINT USING "+###";–500	–500
PRINT USING "###–";500	500
PRINT USING "###–";–500	500–
PRINT USING "!";"SAM"	S
PRINT USING "% %";"SAMANTHA"	SAMAN

MUSIC-SOUND

☆ = ATARI BASIC ONLY ★ = MICROSOFT BASIC ONLY

SOUND speakerno,tone,distortion,volume • *Play a note.*
★SOUND speakerno,tone,distortion,volume [,duration] • *Play a note.*

speakerno = 0, 1, 2, or 3.
tone = 0-255
distortion = 0, 2, 4, 6, 8, 10, 12, 14 (10 for music)
volume = 0-15 (Once per speaker. 8 = Normal)
duration = 0-255 (Increments of 1/60th second)
(Continuous if omitted)

NOTE	OCTAVE 1 VALUE	OCTAVE 2 VALUE	OCTAVE 3 VALUE	HIGH-C
C	243	121	60	29
C#	230	114	57	
D	217	108	53	
D#	204	102	50	
E	193	96	47	
F	182	91	45	
F#	173	85	42	
G	162	81	40	
G#	153	76	37	
A	144	72	35	
A#	136	68	33	
B	128	64	31	

ATARI ERROR CODES

CODE	MEANING
2	*Out of memory.*
3	*Invalid range, or, negative value not allowed.*
4	*Max 128 variable names has been exceeded.*
5	*String too long for DIMension.*
6	*READing beyond end of DATA.*
7	*Value greater than 32767 or not positive.*
8	*Invalid INPUT data type (num/char).*
9	*DIM too large, invalid subscript, DIM not present, or, trying to reDIMension an existing DIM.*
10	*Expression too large, too many GOSUBs, or, you are RETURNing improperly from a GOSUB. (Stack overflow).*
11	*Float-point error. Divided by zero or number too large/small.*
12	*Referenced a non-existing line number.*
13	*NEXT without FOR.*
14	*Statement too complex for BASIC.*
15	*NEXT without FOR. RETURN without GOSUB.*
16	*RETURN without GOSUB.*
17	*Memory error. (Possibly POKEd where you shouldn't)*
18	*String invalid, or, not numeric for VAL statement.*
19	*Program too big for memory.*
20	*Invalid device number.*
21	*Tried to LOAD an invalid LOAD file.*

MICROSOFT ERROR CODES

CODE	MEANING
1	NEXT without FOR.
2	*Syntax error. Punctuation, spelling, datatype, etc.*
3	*RETURN without GOSUB.*
4	*READ cannot find any more DATA, or, not enough INPUT data.*
5	*Invalid parameter specified for a function.*
6	*Overflow or underflow. Result is too large or small.*
7	*Out of memory.*
8	*Referenced a non-existent line number.*
9	*Invalid subscript for a DIM variable.*
10	*ReDIMensioning a variable already DIMensioned.*
11	*Divided by zero.*
12	*Used BASIC-only stmt in command or direct mode.*
13	*Wrong data-type used. (String, num, etc.)*
15	*Value too large. String exceeds 255 characters.*
16	*Statement too long or complex.*
17	*CONT is not possible at this time.*
18	*User function invalid or not defined.*
19	*Program ended during ON ERROR processing. No RESUME.*
20	*RESUME without ON ERROR.*
21	*FOR without NEXT.*

STATUS CODES

CODE	MEANING
1	*No errors.*
3	*End-of-file.*
128	*BREAK key pressed.*
129	*Tried to OPEN a device already open.*
130	*Device specified does not exist.*
131	*Tried to READ a write-only device (printer, etc).*
132	*Illegal command or statement.*
133	*File not OPENed.*
134	*Invalid device number.*
135	*Tried to WRITE to a read-only file.*
136	*End-of-file reached.*
137	*Record too large. Greater than 256 bytes.*
138	*Unable to access device.*
139	*Bad data or device error.*
140	*Input error on serial bus.*
141	*Cursor not in range for this mode.*
142	*Data overrun on serial bus.*
143	*Checksum error on serial bus data.*
144	*Trying to write on a write-protected disk.*
145	*Bad disk write, or, bad screen write.*
146	*Function not available for device.*
147	*Not enough RAM for this graphics mode.*
160	*Invalid drive number specified.*
161	*Maximum open files exceeded.*
162	*Disk capacity exceeded.*
163	*Input/output error on device.*
164	*Cannot access file. Possible bad directory.*
165	*Invalid file name specified.*
166	*Length of POINT data invalid.*
167	*File is 'locked'.*
168	*Invalid command.*
169	*Maximum entries (64) are in the directory.*
170	*File specified not found.*
171	*Invalid POINT.*

BASIC FUNCTIONS

☆ = ATARI BASIC ONLY ★ = MICROSOFT BASIC ONLY

FUNCTION	RETURNS
CHARACTER/ STRING FUNCTIONS	
CHR$(num)	*Character value of* num. *(65="A")*
★INSTR([num,] a$,b$)	*Position of substring* b$ *within larger* a$. *Scan starts at* num, *or first position if* num *not specified. Returns 0 if* b$ *not found.*
★LEFT$(str,num)	num *characters from left side of* string.
LEN(str)	*Length of* string.
★MID$(str,start [,len])	str *chars from* start *position for length* len. *(Chars from* start *to end if* len *not used).*
★RIGHT$(str,num)	num *characters from right side of* string.
★SCRN$(x,y)	*Character or color-code at* x,y *on screen.*
STR$(num)	*String of* number, *including sign. ("– 123")*
★STRING$(num,char)	*String of* char *repeated* num *times.*

NUMERIC FUNCTIONS

ABS(num)	*Positive value of* number. *(– 3.25 = 3.25)*
ASC(str)	*Value of first char in* string. *("A" = 65)*
INT(num)	num *if* num *is whole number. Next lowest whole number if a decimal. (3.25 = 3) (– 3.25 = – 4)*
☆INT(RND(1) * range)+ bottom	*Random whole number between* bottom *minus one* and bottom *plus* range. INT(RND(1) * 100)+30 = Number 30 to 129.
☆RND(anynum)	*Random number between 0 and 1.*
★RND [(0)]	*Random number between 0 and 1.*
★RND(whole num)	*Random number between 0 and* whole num.
SGN(num)	*0 (Zero), 1 (Positive), – 1 (Negative).*
VAL(str)	*Number from a* string *starting with a number.*
★VAL(str)	*Number extracted from* string *(ignores blanks). If first* string *char = non-number, returns 0.*

BASIC FUNCTIONS

☆ = ATARI BASIC ONLY ★ = MICROSOFT BASIC ONLY

FUNCTION	RETURNS
MATH FUNCTIONS	
ATN(num)	*Angle whose* TANGENT *is* num.
COS(angle)	*Cosine of* angle.
☆CLOG(num)	*Logarithm of* num *to the base 10.*
☆DEG	*Forces trig function use of degrees.*
EXP(num)	*Inverse* LOG *of* num.
LOG(num)	*Natural logarithm of* num.
☆RAD	*Forces trig function use of radians(default).*
SIN(angle)	*Sine of* angle.
SQR(+num)	*Square root of positive* num.
★TAN(angle)	*Tangent of* angle.

Microsoft BASIC operates only in RADIANS. Atari — See DEG and RAD above.

Radians=Degrees/ 57.29577951 Degrees=Radians * 57.29577951

MEMORY FUNCTIONS

☆ADR(var$)	*Memory address of string* variable.
★ERL	BASIC *error line number.*
★ERR	*System error code number.*
FRE(0)	*Number of bytes still unused by* BASIC.
PEEK(addr)	*Value of byte at* addr.
★STACK	*Number of available time slots left in the computer's time parameter stack. (Max = 20) (1 for each* AFTER *used, 1 for each* SOUND *dur).*
★TIME	*Internal computer time (6 digits).*
★TIME$	*Wall clock time. Hours, mins, secs.* HH:MM:SS
☆USR(addr [,parm, . . .])	*Results of mach lang rtn at* addr. parm*(s) are parameters from* BASIC *to the rtn.*
USR(addr [,parm])	*Results of mach lang rtn at* addr. parm = *1 parameter allowed.*
★VARPTR(var)	*Memory address of* variable. *Can use* PLMx, CHRx, *and* RESERVE *as* var. *(From* OPTION)
★userfunct(arg)	*Results of* DEFined *user function.*

INPUT/ OUTPUT FUNCTIONS

★EOF(iocb-num)	0 = *End of file. Non-zero = Not end of file.*
★INKEY$	*Value of keyboard key press. Blanks not gotten.*
☆PADDLE(num 0-7)	*Position of paddle, 0 counterclockwise to 128.*
☆PTRIG(num 0-7)	*0 = Paddle trigger pressed. 1 = No press.*
★STATUS(iocb-num)	*System error number.*
★STATUS("filespec")	*System error number.*
☆STICK(joy-num 0-3)	*Position of joystick 1-4. 14 = North, 13 = South, 7 = East, 11 = West, 6 = NE, 10 = NW, 5 = SE, 9 = SW, 15 = Center.*
☆STRIG(joy-num 0-3)	*0 = Joystick trigger pressed. 1 = No press.*

PADDLE, JOYSTICK, KEYBOARD PEEKS

PEEK(padpos)	*1-228, Paddle position. (padpos = 624-631).*
PEEK(padtrig)	*Paddle trigger-press (0=Press, 1=No Press). (padtrig = 636-643).*
PEEK(joypos)	*Joystick position (see* STICK, *above). (joypos = 632-635).*
PEEK(joytrig)	*Joystick trigger-press (0=Press, 1=No Press). (joytrig = 644-647)*
PEEK(764)	*Value of last keypress. (255=none)*
PEEK(53279)	*Right-panel key-press value. 3 =* OPTION KEY, *5 =* SELECT KEY, *6 =* START KEY, *7 =* None

DEVICE CODES & IOCBs

A device code is used as part of a file specification to identify the device to which the data is read or written. Sometimes the device code alone is used as a specification, depending upon the command.

The filespec is usually written as: "dev:filename.ext"

C:	**Cassette**
D1:	**Disk drive 1**
D2:	**Disk drive 2**
D3:	**Disk drive 3**
D4:	**Disk drive 4**
E:	**Screen editor (Screen & Keyboard)**
K:	**Keyboard**
P:	**Printer**
R:	**RS-232 device**
S:	**Screen (Input and Output)**

An IOCB (Input-Output Control Block) is a block of information which is used by the computer to keep track of your activity with a device. For example, it must remember which record you read last, and it must know which device you are using.

There is room for 8 IOCBs (0-7) on your computer chip. The system uses IOCB #0 for Screen Editor processing, IOCB #6 for the GRAPHICS processing, and IOCB #7 for LPRINT, CLOAD, and CSAVE processing.

The IOCB is really the device or device number after it is assigned by the OPEN statement. Whenever you see instructions showing iocb as a parameter, this means you must use an IOCB number in its place.

You may freely use IOCBs 1 thru 5, as they are not used by the system. Remember that CLOSEing a file releases the IOCB for use again, so that you do not exceed the number of available I/O Control Blocks.

RESERVED WORDS

☆ = ATARI BASIC ONLY ★ = MICROSOFT BASIC ONLY

ABS	☆DEG	LEN	☆POP	STEP
☆ADR	★DEL	LET	☆POSITION	☆STICK
★AFTER	DIM	★LINE	PRINT	☆STRIG
AND	DOS	LIST	☆PTRIG	STOP
ASC	☆DRAWTO	LOAD	PUT	STR$
★AT	★ELSE	☆LOCATE	☆RAD	★STRING$
ATN	END	★LOCK	★RANDOMIZE	★TAB
★AUTO	☆ENTER	LOG	READ	★TAN
★BASE	★EOF	☆LPRINT	REM	THEN
☆BYE	★ERL	★MERGE	★RENUM	★TIME
★CHR	★ERR	★MID$	★RESERVE	★TIME$
CHR$	★ERROR	★MOVE	RESTORE	TO
★CLEAR	EXP	★NAME	★RESUME	☆TRAP
CLOAD	★FILL	NEW	RETURN	★TROFF
☆CLOG	FOR	NEXT	★RIGHT$	★TRON
CLOSE	FRE	NOT	RND	★UNLOCK
☆CLR	GET	NOTE	RUN	★USING
★CLS	GOSUB	ON	SAVE	USR
COLOR	GOTO	OPEN	★SCRN$	VAL
☆COM	GRAPHICS	★OPTION	SETCOLOR	★VARPTR
★COMMON	IF	OR	SGN	★VERIFY
CONT	★INKEY$	☆PADDLE	SIN	★WAIT
COS	INPUT	PEEK	SOUND	☆XIO
CSAVE	★INSTR	★PLM	★SPC	★XOR
DATA	INT	PLOT	SQR	
★DEF	★KILL	☆POINT	★STACK	
	★LEFT$	POKE	STATUS	

*** = Split screen** — **SCREEN POKE TABLE**

MODES CLR/ NOCLR	POKE ADDRESS --RANGE--	NUMBER OF SCREEN LINES	NUMBER OF SCREEN POSIT	NUMBER OF SCREEN POKES	NUMBER OF LINE POSIT	NUMBER OF LINE POKES	POS PER POKE
0/32	15424-16383	24	960	960	40	40	1
16/48	15424-16383	24	960	960	40	40	1
1/33 *	15744-16143	20	400	400	20	20	1
17/49	15744-16223	24	480	480	20	20	1
2/34 *	15964-16163	10	200	200	20	20	1
18/50	15964-16203	12	240	240	20	20	1
3/35 *	15984-16183	20	800	200	40	10	4
19/51	15984-16223	24	960	240	40	10	4
4/36 *	15744-16143	40	3200	400	80	10	8
20/52	15744-16223	48	3840	480	80	10	8
5/37 *	15264-16063	40	3200	800	80	20	4
21/53	15264-16223	48	3840	960	80	20	4
6/38 *	14304-15903	80	12800	1600	160	20	8
22/54	14304-16223	96	15360	1920	160	20	8
7/39 *	12384-15583	80	12800	3200	160	40	4
23/55	12384-16223	96	15360	3840	160	40	4
8/40 *	8528-14927	160	51200	6400	320	40	8
24/56	8528-16207	192	61440	7680	320	40	8
9/41	8528-16207	192	15360	7680	80	40	2
25/57	8528-16207	192	15360	7680	80	40	2
10/42	8528-16207	192	15360	7680	80	40	2
26/58	8528-16207	192	15360	7680	80	40	2
11/43	8528-16207	192	15360	7680	80	40	2
27/59	8528-16207	192	15360	7680	80	40	2

- To find screen addresses above, enter the graphics mode desired, then: **PRINT PEEK(88)+256*PEEK(89)**
- In split screen mode, bottom screen is at 16244-16384, 160 positions. To find its start: **PRINT PEEK(660)+256*PEEK(661)**
- It may be necessary to use the two above items if your **ATARI** has more than 16K memory, as the actual screen addresses may vary.
- As you can see, the screen addresses overlap, so, when you move from one **GRAPHICS** mode to another, you may see data left over from another screen unless you **clear the area first.**

SPECIAL KEYBOARD KEYS

KEY	FUNCTION
)(	Atari key. Toggles normal/reverse graphics.
BACK S	Cursor left. Replaces left character with a space.
BREAK	Interrupts execution until **CONT** is entered.
CAPS	Sets screen to upper-case graphics mode.
CLEAR	Clears screen, homes cursor (moves it to upper left).
CTRL ↑	Cursor up.
CTRL ↓	Cursor down.
CTRL →	Cursor right.
CTRL ←	Cursor left.
CTRL 1	Stops/restarts screen display.
CTRL 2	Rings buzzer.
CTRL 3	End-of-file.
DELETE	Deletes line.
CTRL DELETE	Deletes character.
ESC	Used in a string during **PRINT**, precedes cursor/screen controls within **BASIC** program.
INSERT	Inserts a line.
CTRL INSERT	Inserts a character.
LOWR	Sets screen to lower-case graphics mode.
RETURN	Enters data.
TAB	Tabs to next position.
SHIFT TAB	Sets **TAB.** (Defaults = 7,15,23,31,39)
CTRL TAB	Clears **TAB.**
SYSTEM RESET	Halts execution, clears vars, margins, screens, graphics mode 0.

COLOR REGISTER USAGE

SETCOLOR codes for Registers 0-4 (4-8 in Microsoft **BASIC**):

COLOR	RANGE	COLOR	RANGE
0	BLACK to WHITE	8	BLUE to SKY-BLUE
1	BROWN to YELLOW	9	DULL-BLUE to AQUA
2	RED to GOLD	10	AQUA to BLUE-GREEN
3	RED to OFF-PINK	11	YELLOW-GREEN to LT-GREEN
4	RED to PINK	12	PEA-GREEN to YELLOW-GREEN
5	MAGENTA to VIOLET	13	OLIVE-GREEN to CHARTREUSE
6	PURPLE to LT-PURPLE	14	ORANGE to GOLD
7	BLUE to LT-BLUE	15	RED-ORANGE to OFF-PINK

See "**SET PLAYER/MISSILE COLOR**" panel for **P/M** Register color codes.

CONVENTIONS USED HERE

Because the registers used by the two different **BASICS** vary, below is a chart showing our notation for these panels. To set color in a register, use **SETCOLOR** or **POKE**.

NANOS REG NAME	ATARI SETC REG	MSOFT SETC REG	COLOR REG ADDR
A	none	0	704
B	none	1	705
C	none	2	706
D	none	3	707
0	0	4	708
1	1	5	709
2	2	6	710
3	3	7	711
4	4	8	712

GENERAL INFORMATION

- Modes 1 thru 8 are split-screen modes. The **MODE 0 SCREEN** becomes their lower screen. All specs and rules for **MODE 0** apply to lower screens.
- To print on upper-screens, use: **PRINT#6** ...(Upper-screen device)
- Modes 16-27 are full-screen versions of 0-11.
- Modes 32-43 are same as 0-11, but do not clear screen when invoked.
- Modes 48-59 are same as 16-27, but do not clear screen when invoked.
- **COLOR** stmt in modes 0-2 specify character for **PLOT/DRAWTO**. (See 0-255 codes panel for registers used per character).
- **COLOR** stmt in modes 3-11 specify codes used for **PLOT/DRAWTO**.
- Modes 9-11 work in computers with **GTIA** chip. Early versions may not.
- In modes 3-11, printed/plotted ctl chars produce actions, not colors.
- When changing modes, color regs are reset to defaults, so be sure to reset them when required.

GRAPHICS MODE 0

- Registers 0 and 3 not used.
- Register 1 ignores color, sets character luminance only.
- Register 2 sets screen and character color, and screen luminance.
- Register 4 sets border color/lum.

GRAPHICS MODES 1 AND 2

- Registers 0-3 set color/lum for all characters. (See 0-255 codes panel)
- Register 4 sets color/lum for upper (border) screen.

GRAPHICS MODES 3, 5, AND 7

- Register 0 sets color/lum for **POKE** bit-pair value 01, and for **PRINT#6**, **PLOT**, and **DRAWTO**, sets color/lum for value 1 and every 4th value after.
- Register 1 sets color/lum for **POKE** bit-pair value 10, and for **PRINT#6**, **PLOT**, and **DRAWTO**, sets color/lum for value 2 and every 4th value after.
- Register 2 sets color/lum for **POKE** bit-pair value 11, and for **PRINT#6**, **PLOT**, and **DRAWTO**, sets color/lum for value 3 and every 4th value after.
- Register 3 is not used.
- Register 4 sets color/lum for **POKE** bit-pair value 00, and for **PRINT#6**, **PLOT**, and **DRAWTO**, sets color/lum for value 0 and every 4th value after. It also sets color/lum for upper (border) screen, and so, characters using this register also appear invisible.

GRAPHICS MODES 4 AND 6

- Register 0 sets color/lum for any 1-bit (**POKE**), and for all odd-numbered characters (**PRINT#6**, **PLOT**, **DRAWTO**).
- Registers 1-3 not used.
- Register 4 sets color/lum for any 0-bit (**POKE**), and for all even-numbered characters (**PRINT#6**, **PLOT**, **DRAWTO**), and for upper (border) screen, thus making characters using this register invisible.

GRAPHICS MODE 8

- Registers 0 and 3 not used.
- Register 1 ignores color, sets character luminance for any 1-bit (**POKE**), and for any odd-numbered character (**PRINT#6**, **PLOT**, **DRAWTO**).
- Register 2 sets screen and character color, and screen luminance.
- Register 4 sets color/lum of border.
- Depending upon screen color, **POKE**ing can produce color effects as follows: bit-pair value 01 = Green, 10 = Blue. 11 = White, 00 = None.

GRAPHICS MODE 9

- Registers 0-3 not used.
- Register 4 sets screen and character color, and screen luminance.
- A 1-bit turns on a screen point, 0-bit off when **POKE**ing.
- **PRINT#6**, **PLOT**, **DRAWTO** — Each char controls its own luminance, 0-15, and same for increments of 16 following (16-31, 32-47, etc). (Control cursor to prevent error 141).

GRAPHICS MODE 10

- Each color repeats every 16 numbers up to 255.
- Register 0 sets color/lum for values 4 and 12.
- Register 1 sets color/lum for values 5 and 13.
- Register 2 sets color/lum for values 6 and 14.
- Register 3 sets color/lum for values 7 and 15.
- Register 4 sets color/lum for values 8 thru 11.
- Register A sets color/lum for value 0 and screen, making 0 invisible.
- Register B sets color/lum for value 1.
- Register C sets color/lum for value 2.
- Register D sets color/lum for value 3.
- **POKE**, **PRINT#6**, **PLOT**, and **DRAWTO** all follow these rules. (Control cursor to prevent error 141).

GRAPHICS MODE 11

- Registers 0-3 not used.
- Register 4 sets screen color-only, and character luminance-only.
- **POKE** — Each 4 bits of character determine colors (0-15).
- **PRINT#6**, **PLOT**, **DRAWTO** — Each character controls its own color, beginning with 0-15 and repeating up to 255. (Control cursor to prevent error 141).

Published 1983 by Nanos Systems Corp., P.O. Box 24344, Speedway, IN 46224 (317)244-4078

ISBN 0-915069-19-9

PRINTED IN U.S.A.

DEC	HEX	KEY	MODE 0 PRT	MODE 0 POKE	MODES 1 & 2 PRT/REG	MODES 1 & 2 POK/REG	MODES 1 & 2 PLOT/REG	ASCII
0	00	© ,		Blank	Blank	Blank	Blank	NUL
1	01	©A		!	! 1	! 0	! 1	SOH
2	02	©B		"	" 1	" 0	" 1	STX
3	03	©C		#	# 1	# 0	# 1	ETX
4	04	©D		$	$ 1	$ 0	$ 1	EOT
5	05	©E		%	% 1	% 0	% 1	ENQ
6	06	©F		&	& 1	& 0	& 1	ACK
7	07	©G		'	' 1	' 0	' 1	BEL
8	08	©H		(	(1	(0	(1	BS
9	09	©I		)	) 1	) 0	) 1	HT
10	0A	©J		*	* 1	* 0	* 1	LF
11	0B	©K		+	+ 1	+ 0	+ 1	VT
12	0C	©L		,	, 1	, 0	, 1	FF
13	0D	©M		–	– 1	– 0	– 1	CR
14	0E	©N		.	. 1	. 0	. 1	SO
15	0F	©O		/	/ 1	/ 0	/ 1	SI
16	10	©P		0	0 1	0 0	0 1	DLE
17	11	©Q		1	1 1	1 0	1 1	DC1
18	12	©R		2	2 1	2 0	2 1	DC2
19	13	©S		3	3 1	3 0	3 1	DC3
20	14	©T		4	4 1	4 0	4 1	DC4
21	15	©U		5	5 1	5 0	5 1	NAK
22	16	©V		6	6 1	6 0	6 1	SYN
23	17	©W		7	7 1	7 0	7 1	ETB
24	18	©X		8	8 1	8 0	8 1	CAN
25	19	©Y		9	9 1	9 0	9 1	EM
26	1A	©Z		:	: 1	: 0	: 1	SUB
27	1B	ESC		;	; 1	; 0	; 1	ESC
28	1C	©↑		<	< 1	< 0	< 1	FS
29	1D	©↓	DOWN	=	= 1	= 0	= 1	GS
30	1E	©←	LEFT	>	> 1	> 0	> 1	RS
31	1F	©→	RIGHT	?	? 1	? 0	? 1	US
32	20	Space	Space	@	Space	@ 0	Space	Space
33	21	!	!	A	! 0	A 0	! 0	!
34	22	"	"	B	" 0	B 0	" 0	"
35	23	#	#	C	# 0	C 0	# 0	#
36	24	$	$	D	$ 0	D 0	$ 0	$
37	25	%	%	E	% 0	E 0	% 0	%
38	26	&	&	F	& 0	F 0	& 0	&
39	27	'	'	G	' 0	G 0	' 0	'
40	28	(	(	H	(0	H 0	(0	(
41	29	)	)	I	) 0	I 0	) 0	)
42	2A	*	*	J	* 0	J 0	* 0	*
43	2B	+	+	K	+ 0	K 0	+ 0	+
44	2C	,	,	L	, 0	L 0	, 0	,
45	2D	–	–	M	– 0	M 0	– 0	–
46	2E	.	.	N	. 0	N 0	. 0	.
47	2F	/	/	O	/ 0	O 0	/ 0	/
48	30	0	0	P	0 0	P 0	0 0	0
49	31	1	1	Q	1 0	Q 0	1 0	1
50	32	2	2	R	2 0	R 0	2 0	2
51	33	3	3	S	3 0	S 0	3 0	3
52	34	4	4	T	4 0	T 0	4 0	4
53	35	5	5	U	5 0	U 0	5 0	5
54	36	6	6	V	6 0	V 0	6 0	6
55	37	7	7	W	7 0	W 0	7 0	7
56	38	8	8	X	8 0	X 0	8 0	8
57	39	9	9	Y	9 0	Y 0	9 0	9
58	3A	:	:	Z	: 0	Z 0	: 0	:
59	3B	;	;	[	; 0	[0	; 0	;
60	3C	<	<	\	< 0	\ 0	< 0	<
61	3D	=	=	]	= 0	] 0	= 0	=
62	3E	>	>	^	> 0	^ 0	> 0	>
63	3F	?	?	_	? 0	_ 0	? 0	?

© = CONTROL KEY

DEC	HEX	KEY	MODE 0 PRT	MODE 0 POKE	MODES 1 & 2 PRT/REG	MODES 1 & 2 POK/REG	MODES 1 & 2 PLOT/REG	ASCII
64	40	@	@	♥	@ 0	Space	@ 0	@
65	41	A	A	▗▖	A 0	! 1	A 0	A
66	42	B	B	▝▘	B 0	" 1	B 0	B
67	43	C	C	▄	C 0	# 1	C 0	C
68	44	D	D	▖	D 0	$ 1	D 0	D
69	45	E	E	▗	E 0	% 1	E 0	E
70	46	F	F	▘	F 0	& 1	F 0	F
71	47	G	G	▝	G 0	' 1	G 0	G
72	48	H	H	▟	H 0	(1	H 0	H
73	49	I	I	▙	I 0	) 1	I 0	I
74	4A	J	J	▜	J 0	* 1	J 0	J
75	4B	K	K	▛	K 0	+ 1	K 0	K
76	4C	L	L	▌	L 0	, 1	L 0	L
77	4D	M	M	▐	M 0	- 1	M 0	M
78	4E	N	N	▀	N 0	. 1	N 0	N
79	4F	O	O	▄	O 0	/ 1	O 0	O
80	50	P	P	♠	P 0	0 1	P 0	P
81	51	Q	Q	▗	Q 0	1 1	Q 0	Q
82	52	R	R	▄	R 0	2 1	R 0	R
83	53	S	S	▖▗	S 0	3 1	S 0	S
84	54	T	T	●	T 0	4 1	T 0	T
85	55	U	U	▗	U 0	5 1	U 0	U
86	56	V	V	▖	V 0	6 1	V 0	V
87	57	W	W	▄	W 0	7 1	W 0	W
88	58	X	X	▖▗	X 0	8 1	X 0	X
89	59	Y	Y	▌	Y 0	9 1	Y 0	Y
90	5A	Z	Z	▖	Z 0	: 1	Z 0	Z
91	5B	[	[	E	[0	; 1	[0	[
92	5C	\	\	↑	\ 0	< 1	\ 0	\
93	5D	]	]	↓	] 0	= 1	] 0	]
94	5E	^	^	←	^ 0	> 1	^ 0	^
95	5F	_	_	→	_ 0	? 1	_ 0	_
96	60	©.	♦	♦	@ 1	@ 1	@ 1	'
97	61	a	a	a	A 1	A 1	A 1	a
98	62	b	b	b	B 1	B 1	B 1	b
99	63	c	c	c	C 1	C 1	C 1	c
100	64	d	d	d	D 1	D 1	D 1	d
101	65	e	e	e	E 1	E 1	E 1	e
102	66	f	f	f	F 1	F 1	F 1	f
103	67	g	g	g	G 1	G 1	G 1	g
104	68	h	h	h	H 1	H 1	H 1	h
105	69	i	i	i	I 1	I 1	I 1	i
106	6A	j	j	j	J 1	J 1	J 1	j
107	6B	k	k	k	K 1	K 1	K 1	k
108	6C	l	l	l	L 1	L 1	L 1	l
109	6D	m	m	M	M 1	M 1	M 1	m
110	6E	n	n	n	N 1	N 1	N 1	n
111	6F	o	o	o	O 1	O 1	O 1	o
112	70	p	p	p	P 1	P 1	P 1	p
113	71	q	q	q	Q 1	Q 1	Q 1	q
114	72	r	r	r	R 1	R 1	R 1	r
115	73	s	s	s	S 1	S 1	S 1	s
116	74	t	t	t	T 1	T 1	T 1	t
117	75	u	u	u	U 1	U 1	U 1	u
118	76	v	v	v	V 1	V 1	V 1	v
119	77	w	w	w	W 1	W 1	W 1	w
120	78	x	x	x	X 1	X 1	X 1	x
121	79	y	y	y	Y 1	Y 1	Y 1	y
122	7A	z	z	z	Z 1	Z 1	Z 1	z
123	7B	©;	♠	♠	[1	[1	[1	{
124	7C	|	|	|	\ 1	\ 1	\ 1	|
125	7D	CLEAR		↖	CLEAR	] 1	CLEAR	}
126	7E	BACKS		◀	^ 1	^ 1	^ 1	~
127	7F	TAB		▶	_ 1	_ 1	_ 1	DEL

© = CONTROL KEY

DEC	HEX	KEY	MODE 0 PRT	MODE 0 POKE	MODES 1 & 2 PRT/REG	MODES 1 & 2 POK/REG	MODES 1 & 2 PLOT/REG
128	80	)|(© ,			Space	Space	Space
129	81	)|(© A			! 3	! 2	! 3
130	82	)|(© B			” 3	” 2	” 3
131	83	)|(© C			# 3	# 2	# 3
132	84	)|(© D			$ 3	$ 2	$ 3
133	85	)|(© E			% 3	% 2	% 3
134	86	)|(© F			& 3	& 2	& 3
135	87	)|(© G			’ 3	’ 2	’ 3
136	88	)|(© H			(3	(2	(3
137	89	)|(© I			) 3	) 2	) 3
138	8A	)|(© J			* 3	* 2	* 3
139	8B	)|(© K			+ 3	+ 2	+ 3
140	8C	)|(© L			, 3	, 2	, 3
141	8D	)|(© M			– 3	– 2	– 3
142	8E	)|(© N			. 3	. 2	. 3
143	8F	)|(© O			/ 3	/ 2	/ 3
144	90	)|(© P			0 3	0 2	0 3
145	91	)|(© Q			1 3	1 2	1 3
146	92	)|(© R			2 3	2 2	2 3
147	93	)|(© S			3 3	3 2	3 3
148	94	)|(© T			4 3	4 2	4 3
149	95	)|(© U			5 3	5 2	5 3
150	96	)|(© V			6 3	6 2	6 3
151	97	)|(© W			7 3	7 2	7 3
152	98	)|(© X			8 3	8 2	8 3
153	99	)|(© Y			9 3	9 2	9 3
154	9A	)|(© Z			: 3	: 2	: 3
155	9B	RETURN			RETURN	; 2	RETURN
156	9C	DELETE line			< 3	< 2	< 3
157	9D	INSERT line			= 3	= 2	= 3
158	9E	CLR TAB			> 3	> 2	> 3
159	9F	SET TAB			? 3	? 2	? 3
160	A0	)|(Space				@ 2	2
161	A1	)|(!			! 2	A 2	! 2
162	A2	)|(”			” 2	B 2	” 2
163	A3	)|(#			# 2	C 2	# 2
164	A4	)|($			$ 2	D 2	$ 2
165	A5	)|(%			% 2	E 2	% 2
166	A6	)|(&			& 2	F 2	& 2
167	A7	)|(’			’ 2	G 2	’ 2
168	A8	)|((			(2	H 2	(2
169	A9	)|( )			) 2	I 2	) 2
170	AA	)|(*			* 2	J 2	* 2
171	AB	)|(+			+ 2	K 2	+ 2
172	AC	)|(,			, 2	L 2	, 2
173	AD	)|(–			– 2	M 2	– 2
174	AE	)|(.			. 2	N 2	. 2
175	AF	)|(/			/ 2	O 2	/ 2
176	B0	)|(0			0 2	P 2	0 2
177	B1	)|(1			1 2	Q 2	1 2
178	B2	)|(2			2 2	R 2	2 2
179	B3	)|(3			3 2	S 2	3 2
180	B4	)|(4			4 2	T 2	4 2
181	B5	)|(5			5 2	U 2	5 2
182	B6	)|(6			6 2	V 2	6 2
183	B7	)|(7			7 2	W 2	7 2
184	B8	)|(8			8 2	X 2	8 2
185	B9	)|(9			9 2	Y 2	9 2
186	BA	)|(:			: 2	Z 2	: 2
187	BB	)|(;			; 2	[2	; 2
188	BC	)|(<			< 2	\ 2	< 2
189	BD	)|(=			= 2	] 2	= 2
190	BE	)|(>			> 2	^ 2	> 2
191	BF	)|(?			? 2	_ 2	? 2

© = CONTROL KEY

)|(= ATARI KEY, REVERSE GRAPHICS

DEC	HEX	KEY	MODE 0 PRT	MODE 0 POKE	MODES 1 & 2 PRT/REG	MODES 1 & 2 POK/REG	MODES 1 & 2 PLOT/REG
192	C0	)|(@	@	♥	@ 2	3	@ 2
193	C1	)|(A	A	┼	A 2	! 3	A 2
194	C2	)|(B	B	▀	B 2	" 3	B 2
195	C3	)|(C	C	┴	C 2	# 3	C 2
196	C4	)|(D	D	┤	D 2	$ 3	D 2
197	C5	)|(E	E	┐	E 2	% 3	E 2
198	C6	)|(F	F	╱	F 2	& 3	F 2
199	C7	)|(G	G	╲	G 2	' 3	G 2
200	C8	)|(H	H	◢	H 2	(3	H 2
201	C9	)|(I	I	▗	I 2	) 3	I 2
202	CA	)|(J	J	◣	J 2	* 3	J 2
203	CB	)|(K	K	▖	K 2	+ 3	K 2
204	CC	)|(L	L	▟	L 2	, 3	L 2
205	CD	)|(M	M	▀	M 2	- 3	M 2
206	CE	)|(N	N	▙	N 2	. 3	N 2
207	CF	)|(O	O	▗	O 2	/ 3	O 2
208	D0	)|(P	P	✥	P 2	0 3	P 2
209	D1	)|(Q	Q	▗	Q 2	1 3	Q 2
210	D2	)|(R	R	▀	R 2	2 3	R 2
211	D3	)|(S	S	▀	S 2	3 3	S 2
212	D4	)|(T	T	○	T 2	4 3	T 2
213	D5	)|(U	U	▀	U 2	5 3	U 2
214	D6	)|(V	V	▀	V 2	6 3	V 2
215	D7	)|(W	W	▀	W 2	7 3	W 2
216	D8	)|(X	X	▀	X 2	8 3	X 2
217	D9	)|(Y	Y	▀	Y 2	9 3	Y 2
218	DA	)|(Z	Z	▗	Z 2	: 3	Z 2
219	DB	)|([	[	▗	[2	; 3	[2
220	DC	)|(\	\	▗	\ 2	< 3	\ 2
221	DD	)|( ]	]	▗	] 2	= 3	] 2
222	DE	)|(^	^	▗	^ 2	> 3	^ 2
223	DF	)|(_	_	▗	_ 2	? 3	_ 2
224	E0	)|(©.	♥		@ 3	@ 3	@ 3
225	E1	)|(a	a		A 3	A 3	A 3
226	E2	)|(b	b		B 3	B 3	B 3
227	E3	)|(c	c		C 3	C 3	C 3
228	E4	)|(d	d		D 3	D 3	D 3
229	E5	)|(e	e		E 3	E 3	E 3
230	E6	)|(f	f		F 3	F 3	F 3
231	E7	)|(g	g		G 3	G 3	G 3
232	E8	)|(h	h		H 3	H 3	H 3
233	E9	)|(i	i		I 3	I 3	I 3
234	EA	)|(j	j		J 3	J 3	J 3
235	EB	)|(k	k		K 3	K 3	K 3
236	EC	)|(l	l		L 3	L 3	L 3
237	ED	)|(m	m		M 3	M 3	M 3
238	EE	)|(n	n		N 3	N 3	N 3
239	EF	)|(o	o		O 3	O 3	O 3
240	F0	)|(p	p		P 3	P 3	P 3
241	F1	)|(q	q		Q 3	Q 3	Q 3
242	F2	)|(r	r		R 3	R 3	R 3
243	F3	)|(s	s		S 3	S 3	S 3
244	F4	)|(t	t		T 3	T 3	T 3
245	F5	)|(u	u		U 3	U 3	U 3
246	F6	)|(v	v		V 3	V 3	V 3
247	E7	)|(w	w		W 3	W 3	W 3
248	F8	)|(x	x		X 3	X 3	X 3
249	F9	)|(y	y		Y 3	Y 3	Y 3
250	FA	)|(z	z		Z 3	Z 3	Z 3
251	FB	)|(©;	♣		[3	[3	[3
252	EC	)|(|	▌	▐	\ 3	\ 3	\ 3
253	FD	©2	BELL	↖	] 3	] 3	] 3
254	FE	©DEL		◀	^ 3	^ 3	^ 3
255	FF	©INS		▶	_ 3	_ 3	_ 3

© = CONTROL KEY

)\|(= ATARI KEY, REVERSE GRAPHICS

PROGRAMMING PLAYER/MISSILE GRAPHICS

Player-missile graphics can also be done in **ATARI BASIC** using **POKE**. In addition to the **MICROSOFT** statements, some **POKE**ing is required. Below is a summary for using Player/Missile graphics.

RESERVE MEMORY FOR PLAYER/MISSILE TABLE

The **P/M** table contains player/missile graphics inserted by you.
In Microsoft BASIC, OPTION PLM1 or OPTION PLM2 does this for you.

In Atari BASIC, the following is used:

A = PEEK(106)—num • Determine the location for the **PM** table.
(Choose largest num of **GRAPHICS MODE** used)

GRAPHICS MODE	DOUBLE-RES num	SINGLE-RES num
0-4	8	16
5	12	16
6	16	24
7	24	32
8-11	36	40

Addr 106 = **BASIC** pgm storage limit.

POKE 106,A • Reset the **BASIC** pgm storage limit to **PM** table addr.

POKE 54279,A • Set **PM** pointer to **PM** table.

SET PM SCREEN RESOLUTION MODE

—**REQUIRED FOR BOTH BASIC LANGUAGES**—

POKE 559,num • **62** (Single-res) **46** (Double-res) Screen 2 (**Normal**)
61 (Single-res) **45** (Double-res) Screen 1 (**Narrow**)
63 (Single-res) **47** (Double-res) Screen 3 (**Wide**)
60 (Single-res) **44** (Double-res) Screen 0 (**Border**)

TURN ON PM GRAPHICS

—**REQUIRED FOR BOTH BASIC LANGUAGES**—

POKE 53277,num • **0** = Turn graphics off. **1** = Turn on **MISSILES** only.
2 = Turn on **PLAYERS** only. **3** = Turn on both.
(If you are going to use a **5th Player** and you want a separate color, **add 16** to above number).

SET SCREEN GRAPHICS PRIORITY

—**REQUIRED FOR BOTH BASIC LANGUAGES**—

POKE 623,num • **1** = Play/mis. **2** = Play/mis 0/1, screen, play/mis 2/3.
4 = Screen. **8** = Screen 0/1, play/mis, screen 2/3.

SET PLAYER/MISSILE COLOR

In Microsoft BASIC, use the SETCOLOR stmt to set player/missile colors.
In Atari BASIC: **POKE addr,color** • **addr** = **704-707** for player/missiles 0-3.

0=Grey	**16**=Gold	**32**=Orange	**48**=Red
64=Pink	**80**=Violet	**96**=Purple	**112**=Blue
128=Blue	**144**=Lt Blue	**160**=Aqua	**176**=Blue-Grn
192=Green	**208**=Yel-Grn	**224**=Org-Grn	**240**=Lt Org

ADD 0-15 TO ABOVE CODES FOR LUMINESCENCE.

SET PLAYER/MISSILE SIZE

—**REQUIRED FOR BOTH BASIC LANGUAGES**—

The **vertical size** of a player/missile is determined by how many values are poked into its area in the **P/M** table. 1 **POKE** = 1 line.

The **width** of a player/missile can be normal, double, or quadruple.
POKE addr,num • For players 0-3, **addr** = **53256-53259**.
num = **0/2**(Normal), **1**(Double), **3**(Quadruple).

POKE 53260,value • For missiles use the following values. When setting width for multiple missiles, add the values together before **POKE**ing.

MISSILE 0	1	2	3	SIZE
0/2	0/8	0/32	0/128	Normal
1	4	16	64	Double
3	12	48	192	Quadruple.

PLAYER/MISSILE TABLES

Obtain the address of the **P/M table** so you can address each area.
In Microsoft **BASIC** enter: **B = VARPTR(PLM1)** -or- **B = VARPTR(PLM2)**
In Atari **BASIC**, enter: **B = 256 * A** (For **A** see **RESERVE MEMORY panel**)

Single-res mode:

range = vertical screen visibility = **32-223**
Missile area = 256 bytes. Each Player area = 256 bytes.

Using **G** to point to the **P/M** screen area:	**G** = B + 768
Using **M** to point to Missile area:	**M** = B + 768 + **range**
Using **P0** to point to the Player 0 area:	**P0** = B + 1024 + **range**
Using **P1** to point to the Player 1 area:	**P1** = B + 1280 + **range**
Using **P2** to point to the Player 2 area:	**P2** = B + 1536 + **range**
Using **P3** to point to the Player 3 area:	**P3** = B + 1792 + **range**
Using **T** to point to the end of the table:	**T** = B + 2047

Double-res mode:

range = vertical screen visibility = **16-111**
Missile area = 128 bytes. Each Player area = 128 bytes.

Using **G** to point to the **P/M** screen area:	**G** = B + 384
Using **M** to point to the Missile area:	**M** = B + 384 + **range**
Using **P0** to point to the Player 0 area:	**P0** = B + 512 + **range**
Using **P1** to point to the Player 1 area:	**P1** = B + 640 + **range**
Using **P2** to point to the Player 2 area:	**P2** = B + 768 + **range**
Using **P3** to point to the Player 3 area:	**P3** = B + 896 + **range**
Using **T** to point to the end of the table:	**T** = B + 1023

When starting, the table has **garbage** in it. Whenever you **POKE** or **MOVE** data around, **it stays there.** You must **clear out what you do not want.**

In Microsoft **BASIC**, the **OPTION PLMx** stmt clears the table for you.
In Atari **BASIC**, you must clear it: **For J = G TO T: POKE J,0: NEXT J**

PLAYER/MISSILE MOTION

Players and Missiles move vertically by **POKE**ing 0 into their old bits and **POKE**ing their shapes back into another location. The vertical visible screen range is noted in the **TABLES** panel.

In Microsoft BASIC the MOVE statement can be used instead of the POKE. It is much more efficient.

Notice in our example of **PLAYER 2** that the top and bottom lines are set to zero. This is so because when you want to move up or down, the zeros always move first, thus clearing out the prior top or bottom lines of the Player at its last position.

To move **PLAYER 2** down 1 line **in Microsoft BASIC:**

```
700 MOVE B+1536+32,B+1536+33,17
```

To move **PLAYER 2** down 1 line **in Atari BASIC:**

```
700 RESTORE          . . . RESTORE THE DATA
800 FOR I = B+1536+33 TO B+1536+33+16
900 READ C: POKE I,C: NEXT I
```

The Atari **BASIC** method is slow, but, the smaller the Player, the faster the movement.

Moving Players and Missiles horizontally is much faster and easier.
There is a horizontal register for each Player and Missile. To move your graphic, simply **POKE** the location into the register.
The width of the screen is from **0-227**. The **visible horizontal area is 41-200.** These are the values you can **POKE**.

POKE a value into 53248 thru 53251 for Players **0-3** respectively.
POKE a value into 53252 thru 53255 for Missiles **3-0** respectively.
For example, to move **PLAYER 2** across the screen and back:

```
1000 FOR I = 41 TO 200: POKE 53250,I: NEXT I
1100 FOR I = 200 TO 41 STEP -1: POKE 53250,I: NEXT I
```

MAKING A MISSILE

The Missile area contains **128** or **256 lines** which correspond to **screen rows** or **lines. The visible range** is noted above for each res mode.
Where you **POKE** a missile determines where it appears on the screen.

Each line in the Missile area is **1 byte long (8 bits)**. Each Missile can be regarded as being **128 or 256 lines high**. If you **POKE** 255 into every Missile line, you get a **great vertical bar** on the screen when you move it into horizontal range. So, a Missile is as long vertically as you desire. If you skip areas, but **POKE** into many different locations, a Missile becomes **fragmented**, or, becomes **multiple Missiles as one**.

Each byte or line dedicates **2 bits to each Missile**.

Missile assigned to 2 bits beneath:	3	2	1	0
Numbers assigned to bits of a byte:	7...6	5...4	3...2	1...0
When bit is on (1), value is:	128..64	32..16	8...4	2...1
Value for Missile when all bits on:	192	48	12	3

So, a Missile can have **4 values**: 0, left value, right value, both values.
The left and right values of a missile turn on the left or right graphic on the screen.

Assuming Single-res mode, to place Missile 3 at top of screen, then
POKE B+768+32,192 (You can also **POKE** in **64** or **128**)
Missile 3 is now in the proper vertical position. **POKE 53255,41** will move it to the proper **horizontal** position.

To make the Missile longer vertically, **POKE** the next location with a Missile 3 value. When more than 1 Missile is on a line, or any part of more than 1, **add the values together before the POKE**. For example, to place Missiles 1 and 3 at the top: **POKE B+768+32,195**
Then **POKE 53252,41** and **POKE 53255,61** for horizontal, and they will appear side-by-side.

THE FIFTH PLAYER

A **5th Player** can be created in place of the 4 Missiles.
It is created like the Player graphic, but **POKE**d into the Missile area.

To enable this player to have its own color, **add 16** to the number that is **POKE**d to set **SCREEN GRAPHICS PRIORITY.** (Otherwise it is 4-color)
For example: **POKE 623,1+16**

Then to assign the color to Player 5: **POKE 711,color**
(See **SET PLAYER/MISSILE COLOR** panel for color codes)

For collision detection, any hit on a Missile is a hit on **Player 5.**

Vertical movement is the **same** as for any other Player.
Horizontal movement is achieved however by **POKE**ing the 4 Missile regs.
Remember that these regs, 53252-53255, control Missiles 3-0 in that sequence, so you must **assume 53255 as the left part of Player 5** unless you **POKE** it in backwards.

Horizontal movement will be slightly slower than for the other Players.

When **POKE**ing the horizontal movement regs for Player 5, the addresses **POKE**d into the regs **must allow for 2 positions for each part of the Player.** If you are using **double width**, this must be **4 positions**, and if you are using **quadruple width**, this must be **8 positions**, otherwise Player 5 will cram up on you.

To set width for Player 5:
POKE 53260,0 ...Normal
POKE 53260,85 ...Double
POKE 53260,255 ...Quadruple

MAKING A PLAYER

There are 4 Player areas in the **P/M** table.
Each Player area contains **128** or **256 lines** which correspond to **screen rows** or **lines**. The **visible range** is noted above for each res mode.
Where you **POKE** a Player determines its vertical location on the screen.

Each line in each Player area is **1 byte (8 bits)** long. Each Player can be regarded as being **128 or 256 lines high**. If you **POKE** 255 into every Player byte, you get a **long vertical bar** on the screen when you move them into horizontal viewing range. So, a Player may be as long vertically as desired. When you skip areas and **POKE** into many different Player locations, the Player becomes fragmented, or, becomes **multiple Players controlled as one.**

The limits of a Player design are 8 bits wide and 256 bits long.
For our example, assume a Player size of 8 by 17.
Below is a sample design of a Player.

128	64	32	16	8	4	2	1	DECIMAL VALUE
0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	1	1	195
1	1	0	0	0	0	1	1	195
0	1	1	0	0	1	1	0	102
0	0	1	0	0	1	0	0	36
0	0	1	0	0	1	0	0	36
1	1	1	1	1	1	1	1	255
1	0	0	0	0	0	0	1	129
1	0	1	0	0	1	0	1	165
1	0	0	0	0	0	0	1	129
1	0	0	1	1	0	0	1	153
1	0	0	0	0	0	0	1	129
1	0	1	0	0	1	0	1	165
1	0	1	1	1	1	0	1	189
1	0	0	0	0	0	0	1	129
1	1	1	1	1	1	1	1	255
0	0	0	0	0	0	0	0	0

The **top line** of the grid contains the **value of each bit** when it is turned on (1). To determine the value of the byte (8 bits), **add the values together.** The numbers on the right side are the **decimal values** for each line or byte of this character.

These values **must be poked into the Player area** to make the Player.
Assuming Single-res mode, to place this graphic into the Player 2 area, we could use the following code:

```
100 DATA 0, 195, 195, 102, 36, 36, 255, 129, 165
200 DATA 129, 153, 129, 165, 189, 129, 255, 0
300 FOR I = B + 1536 + 32 TO B + 1536 + 32 + 16
400 READ C: POKE I,C: NEXT I
```

Of course, this is assuming that the rest of the program has been already coded to use **P/M** graphics.

To move Player 2 into the visible horizontal screen position,

POKE 53250,50

PLAYER/MISSILE COLLISIONS

There are 16 collision registers which can be checked. Some of these are used for other things besides collisions as well, so they may be mentioned elsewhere in other contexts.

REG	ATTACKER	TARGET	REG	ATTACKER	TARGET
53248	**MISSILE 0**	SCREEN 0-3	53252	**PLAYER 0**	SCREEN 0-3
53249	**MISSILE 1**	SCREEN 0-3	53253	**PLAYER 1**	SCREEN 0-3
53250	**MISSILE 2**	SCREEN 0-3	53254	**PLAYER 2**	SCREEN 0-3
53251	**MISSILE 3**	SCREEN 0-3	53255	**PLAYER 3**	SCREEN 0-3
53256	**MISSILE 0**	PLAYER 0-3	53260	**PLAYER 0**	PLAYER 0-3
53257	**MISSILE 1**	PLAYER 0-3	53261	**PLAYER 1**	PLAYER 0-3
53258	**MISSILE 2**	PLAYER 0-3	53262	**PLAYER 2**	PLAYER 0-3
53259	**MISSILE 3**	PLAYER 0-3	53263	**PLAYER 3**	PLAYER 0-3

POKE 53278,0 . . . To clear all of the above registers.

Sample code for testing collision register:

```
1200 V = PEEK(reg)
1210 IF V AND value THEN . . . action . . .
```

Screen or Player number	0	1	2	3
value	**1**	**2**	**4**	**8**

To test for multiple collisions, add the values together.

1210 IF V and 10 THEN . . . action . . . (Players 1 & 3)

MEMORY MAP—PAGE ZERO

START ADDR		END ADDR		DESCRIPTION
DEC	HEX	DEC	HEX	
0	$0000	127	$007F	System memory (RAM).
2	$0002	3	$0003	Addr of Cassette boot routine.
8	$0008	8	$0008	Warmstart flag. 255 = Sys Reset, 0 = Normal.
10	$000A	11	$000B	Addr of DOS routine.
12	$000C	13	$000D	Addr of disk boot routine.
14	$000E	15	$000F	Addr of highest memory loc used by BASIC.
17	$0011	17	$0011	**0 = BREAK key pressed.**
18	$0012	20	$0014	**1/60th second clock.**
32	$0020	44	$002C	Zero-page I/O Control Block (IOCB).
65	$0041	65	$0041	Noise flag. (0 = Silent)
77	$004D	77	$004D	Attract-mode (128) flag.
82	$0052	83	$0053	**Left & right margins (Default = 2 & 39).**
84	$0054	84	$0054	**Cursor row (vert) pos, upper screen.**
85	$0055	86	$0056	**Cursor col (horiz) pos, upper screen.**
88	$0058	89	$0059	**Addr of upper left of screen.**
90	$005A	90	$005A	Prior cursor row (vert) pos, upper screen.
91	$005B	92	$005C	Prior cursor col (horiz) pos, upper screen.
93	$005D	93	$005D	Character covered by cursor, upper screen.
96	$0060	96	$0060	**DRAWTO destination row (vert) position.**
97	$0061	98	$0062	**DRAWTO destination col (horiz) position.**
106	$006A	106	$006A	MSB addr of display handler hi-mem (RAMTOP).
128	$0080	175	$00AF	Page zero, BASIC or ASSEMBLER.
128	$0080	129	$0081	Addr of BASIC lo-mem loc (MEMLO).
144	$0090	145	$0091	Addr of BASIC hi-mem loc (MEMTOP).
176	$00B0	202	$00CA	Page zero BASIC or ASSEMBLER free RAM.
186	$00BA	187	$00BB	Line number (hex) of STOP or TRAP.
195	$00C3	195	$00C3	**Current error number.**
201	$00C9	201	$00C9	PRINT tab width (default = 10).
203	$00CB	207	$00CF	Page zero BASIC or ASSEMBLER free RAM.
208	$00D0	209	$00D1	Page zero BASIC free RAM.
210	$00D2	211	$00D3	BASIC or cartridge program.
212	$00D4	217	$00D9	Float point reg 0.
224	$00E0	229	$00E5	Float point reg 1.
251	$00FB	251	$00FB	**0 = Radians, 6 = Degrees.**
252	$00FC	255	$00FF	Page zero memory.

MEMORY MAP—NON-ZERO PAGES

START ADDR		END ADDR		DESCRIPTION
DEC	HEX	DEC	HEX	
256	$0100	511	$01FF	System stack.
512	$0200	831	$033F	System variables.
559	$022F	559	$022F	**Player/Missile screen resolution register.**
564	$0234	565	$0235	**Light pen, horiz & vert values.**
623	$026F	623	$026F	**Player/Missile priority register.**
624	$0270	631	$0277	**Paddle 0-7 position port.**
632	$0278	635	$027B	**Joystick pos / Lightpen button port (0-3).**
636	$027C	643	$0283	**Paddle 0-7 trigger port.**
644	$0284	647	$0287	**Joystick 0-3 trigger port.**
656	$0290	656	$0290	**Cursor row (vert) pos, lower screen.**
657	$0291	658	$0292	**Cursor col (horiz) pos, lower screen.**
660	$0294	661	$0295	**Addr of upper left of lower screen window.**
675	$02A3	689	$02B1	Tab stop map (1 byte ea, 15 tabs).
694	$02B6	694	$02B6	0 = Normal graphics. Not 0 = reverse graphics.
702	$02BE	702	$02BE	128 = CTRL lock. 64 = CAPS lock. 0 = No lock.
704	$02C0	707	$02C3	Player/Missile color registers.
708	$02C4	712	$02C8	Color registers 0-4.
741	$02E5	742	$02E6	Addr of system hi-mem loc (MEMTOP).
743	$02E7	744	$02E8	Addr of system lo-mem loc (MEMLO).
752	$02F0	752	$02F0	**0 = Cursor on. 1 = Cursor off.**
755	$02F3	755	$02F3	Char mode. 1(blank), 2(norm), 4(vert reflect).
756	$02F4	756	$02F4	Char base. 224 = Upper(default). 226 = Lower.
762	$02FA	762	$02FA	Contains last character or point from screen.
764	$02FC	764	$02FC	**Code of last keypress. (255 = Empty)**
765	$02FD	765	$02FD	**XIO graphics fill character.**
766	$02FE	766	$02FE	Display flag. 1 = Display control char.
767	$02FF	767	$02FF	Screen start/stop. 0=Normal. (CTRL 1 key).
794	$031A	831	$033F	System handler addr tbl (3 bytes per handler).

832	$0340	959	$03BF	I/O Control blocks 0-7 (IOCB, 16 bytes each).
960	$03C0	999	$03E7	Buffer for printer.
1000	$03E8	1020	$03FC	Reserved for future use.
1021	$03FD	1151	$047F	Buffer for cassette.
1152	$0480	1405	$057D	BASIC cartridge mem. (Avail. to mach lang)
1406	$057E	1535	$05FF	BASIC float point rtn.
1536	$0600	1791	$06FF	Free (extra) RAM.
1792	$0700	4863	$12FF	DOS file mgmt RAM.
4864	$1300	9855	$267F	DOS System RAM.
9856	$2680	10879	$2A7F	DOS Disk I/O buffers.
10880	$2A80	31754	$7C1E	**BASIC program area.**
31755	$7C1F	32767	$7FFF	Display list and data (variable sizes).
32768	$8000	40959	$9FFF	Cartridge slot B, or RAM.
40960	$A000	49151	$BFFF	Cartridge slot A, or RAM.
49152	$C000	53247	$CFFF	Unused.
53248	$D000	55295	$D7FF	Hardware regs.
53248	$D000	53263	$D00F	Player/Missile collision registers.
53248	$D000	53255	$D007	Player/Missile horizontal position registers.
53256	$D008	53260	$D00C	Player/Missile size registers.
53277	$D01D	53277	$D01D	Player/Missile enable register.
53279	$D01F	53279	$D01F	Console switch byte. 0 = Switch pressed. POKE 53279,0. Bit: 0 = Start, 1 = Select, 2 = Option.
53760	$D200	53791	$D21F	POKEY (Audio, Paddle ctl, KB scan, RND) chip.
53775	$D20F	53775	$D20F	Serial port ctl reg.
54016	$D300	54017	$D301	Joystick read/write PIA ports A and B.
54018	$D302	54019	$D303	Control regs, ports A and B.
54020	$D304	54271	$D3FF	Same format, PIA ports and control regs.
54272	$D400	54303	$D41F	ANTIC (Screen control) chip.
54279	$D407	54279	$D407	Player/Missile table pointer.
55296	$D800	57343	$DFFF	Floating point package ROM.
57344	$E000	65535	$FFFF	System ROM.

ATARI BASIC TOKENS - TYPE 1
(Tokens at beginning of BASIC lines)

WORD	TOKEN	WORD	TOKEN	WORD	TOKEN
?	40	GET	41	POINT	28
assumed LET	54	GOSUB	12	POKE	31
BYE	14	GO TO	11	POP	39
CLOAD	53	GOTO	10	POSITION	45
CLOSE	17	GRAPHICS	43	PRINT	32
CLR	18	IF	7	PUT	42
COLOR	3	INPUT	2	RAD	33
COM	16	LET	6	READ	34
CONT	15	LIST	4	REM	0
CSAVE	52	LOAD	24	RESTORE	35
DATA	1	LOCATE	49	RETURN	36
DEG	19	LPRINT	51	RUN	37
DIM	20	NEW	22	SAVE	25
DOS	46	NEXT	9	SETCOLOR	48
DRAWTO	47	NOTE	27	SOUND	50
END	21	ON	30	STATUS	26
ENTER	5	OPEN	23	STOP	38
FOR	8	PLOT	44	XIO	29

ATARI BASIC TOKENS — TYPE 2
(Tokens in middle of BASIC lines) △ = blank character

WORD	TOKEN	WORD	TOKEN	WORD	TOKEN	WORD	TOKEN
end line	22	>=	31	△AND	42	FRE	73
,	60	>=	49	△GOSUB	24	INT	80
:	20	^	35	△GOTO	23	LEN	66
;	21	*	36	△NOT	40	LOG	75
#	28	+	37	△OR	41	PADDLE	81
=	45	+	53	△STEP	26	PEEK	70
=	46	–	38	△THEN	27	PTRIG	83
=	34	–	54	△TO	25	RND	72
=	52	/	39	ABS	79	SGN	78
<	32	(	43	ADR	67	SIN	71
<	50	(	55	ASC	64	SQR	77
>	33	(	58	ATN	68	STICK	82
>	51	(	59	CHR$	62	STR$	61
<>	30	)	44	CLOG	76	STRIG	84
<>	48	DimVar(	56	COS	69	USR	63
<=	29	DimVar(	57	EXP	74	VAL	65
<=	47						